An ACG View on G-TAG and Its g-Derivation

Laurence Danlos, Aleksandre Maskharashvili, Sylvain Pogodalla

An ACG View on G-TAG and Its g-Derivation*

Laurence Danlos[1,2,3],
Aleksandre Maskharashvili[4,5,6], and
Sylvain Pogodalla[4,5,6]

[1] Université Paris Diderot (Paris 7), Paris, F-75013, France
[2] ALPAGE, INRIA Paris–Rocquencourt, Paris, F-75013, France
[3] Institut Universitaire de France, Paris, F-75005, France
[4] INRIA, Villers-lès-Nancy, F-54600, France
[5] Université de Lorraine, LORIA, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France
[6] CNRS, LORIA, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France
{laurence.danlos,aleksandre.maskharashvili,sylvain.pogodalla}@inria.fr

Abstract. G-TAG is a Tree Adjoining Grammar (TAG) based formalism which was specifically designed for the task of text generation. Contrary to TAG, the derivation structure becomes primary, as pivot between the conceptual representation and the surface form. This is a shared feature with the encoding of TAG into Abstract Categorial Grammars. This paper propose to study G-TAG from an ACG perspective. We rely on the reversibility property of ACG that makes both parsing and generation fall within a common morphism inversion process. Doing so, we show how to overcome some limitations of G-TAG regarding predicative adjunction and how to propose alternative approaches to some phenomena.

1 Overview

Tree-adjoining grammar (TAG) [8], [9] is one of the most extensively studied grammatical formalism as it considered to be a formalism that can encode the most of natural languages. The point is that TAGs produce string languages that are not in general context free, but mildly context-sensitive. The latter fact means that the tree language produced by a TAG, called *derived tree language*, is not a regular tree language and thus it might be problematic to deal with it in certain tasks. However, in TAG, together with the notion of derived tree, there is defined the notion of *derivation tree* that encodes the history of the process of the derivation of a derived tree. It turns out that while a TAG derived tree language generally is not regular, TAG derivation tree language is regular. This fact makes possible to explore the useful properties of regular tree languages on TAG derivation trees [7]. Furthermore, a type-logical account was given to TAG in [5], where TAG was represented as second order ACGs.

G-TAG [3], [13] is a formalism designed for text generation, which makes use of the notions defined within TAG. However, in G-TAG the central notion of derivation tree (called *g-derivation* trees) is different from one that is defined in TAG. The goal of

* This work has been supported by the French agency Agence Nationale de la Recherche (ANR-12-CORD-0004).

the current paper is to show that g-derivation and g-derived trees can be encoded in the ACG framework in the same spirit as TAG is encoded as ACGs, and also to simulate G-TAG text generation process in the confines of the ACG framework.

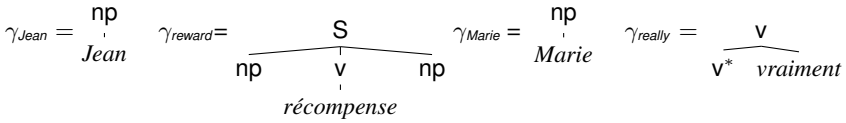
2 Tree Adjoining Grammars

Tree adjoining grammars have been widely used in computational linguistics in a number of tasks including parsing and generation (e.g. [12], [7]).

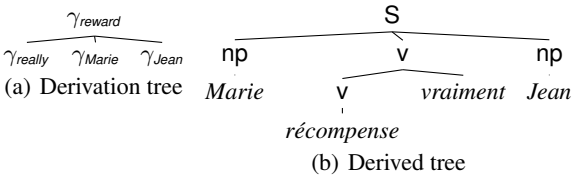
A TAG has two kinds of entries: *initial* and *auxiliary* trees, which are called elementary trees. An initial tree has interior nodes labeled by non-terminal symbol and frontier nodes, i.e. leaves, labeled by the symbols which can be either non-terminal or terminal ones. The non-terminal symbols appearing on the leaves of an initial tree are marked for *substitution*. For instance, γ_{reward}, γ_{John} and γ_{Mary} are initial trees of TAG. Similarly to initial trees, an auxiliary tree has its interior nodes labeled by the non-terminal symbols and frontier nodes labeled by either non-terminal or terminal ones; the frontier non-terminal symbols are marked for substitution as well, *except for one distinguished node* that is called the *foot node* and which has additionally attached an asterisk $*$. The label of the foot node is the same as the root node of the auxiliary tree. For instance, γ_{really} is an auxiliary tree.

A TAG *derived* tree language is produced from elementary trees with the help of *substitution* and *adjunction* operations defined in TAG. In short, TAG substitution is a substitution of frontier non-terminals of an initial tree by other initial trees; TAG adjunction is substitution of a internal node A of an initial tree by an auxiliary tree so that all the daughters of the substituted node become daughters of the node with label A of the substituted auxiliary tree. This process of producing a derived tree is encoded with a *derivation* tree.

For instance, via substituting γ_{Jean} and γ_{Marie} trees in the initial tree γ_{reward} and via adjoining γ_{really} into v node of γ_{reward}, one can get the derived tree given on the figure 1(b)



The process of the production of the derived tree 1(b) is recorded by the corresponding *derivation* tree, which is represented as the tree 1(a).[7]



(a) Derivation tree

(b) Derived tree

Fig. 1. *Marie récompense vraiment Jean*

[7] Here, we omit tree addresses, which are usually used in TAG.

3 ACG Definition

ACGs provide a framework in which several grammatical formalisms may be encoded [6]. ACGs generate languages of linear λ-terms, which generalize both string and tree languages.

Definition 1. *A* higher-order linear signature *is defined to be a triple* $\Sigma = \langle A, C, \tau \rangle$*, where,* A *is a finite set of atomic types (also noted* A_Σ*),* C *is a finite set of constants (also noted* C_Σ*), and* τ *is a mapping from* C *to* $\mathscr{T}_A$ *the set of types built on* A*:* $\mathscr{T}_A ::= A | \mathscr{T}_A \multimap \mathscr{T}_A$ *(also noted* $\mathscr{T}_\Sigma$*). A higher-order linear signature will also be called a* vocabulary*.* $\Lambda(\Sigma)$ *is the set of* λ*-terms built on* Σ*, and for* $t \in \Lambda(\Sigma)$ *and* $\alpha \in \mathscr{T}_\Sigma$ *such that* t *has type* α*, we note* $t :_\Sigma \alpha$

Definition 2. *An* abstract categorial grammar *is a quadruple* $\mathscr{G} = \langle \Sigma, \Xi, \mathscr{L}, s \rangle$ *where:*

1. Σ *and* Ξ *are two higher-order linear signatures, which are called the* abstract vocabulary *and the* object vocabulary*, respectively;*
2. $\mathscr{L} : \Sigma \longrightarrow \Xi$ *is a lexicon from the abstract vocabulary to the object vocabulary. An ACG lexicon* $\mathscr{L}$ *is a homomorphism that maps types and terms built on* Σ *to types and terms built on* Ξ*, in the following way:*
   - *If* $\alpha \multimap \beta \in \mathscr{T}_\Sigma$ *then* $\mathscr{L}(\alpha \multimap \beta) = \mathscr{L}(\alpha) \multimap \mathscr{L}(\beta)$*.*
   - *If* $\lambda x.t, (t\, u) \in \Lambda(\Sigma)$ *then* $\mathscr{L}(\lambda x.t) = \lambda x.\mathscr{L}(t)$ *and* $\mathscr{L}(t\, u) = \mathscr{L}(t)\, \mathscr{L}(u)$
   - *For any constants* $c :_\Sigma \alpha$ *of* Σ *we have* $\mathscr{L}(c) :_\Xi \mathscr{L}(\alpha)$*.*
3. $s \in \mathscr{T}_\Sigma$ *is a type of the abstract vocabulary, which is called the* distinguished type *of the grammar.*

The abstract language *of an ACG* $\mathscr{G} = \langle \Sigma, \Xi, \mathscr{L}, s \rangle$ *is* $\mathcal{A}(\mathscr{G}) = \{t \in \Lambda(\Sigma) \mid t :_\Sigma s\}$
The object language *of the grammar* $\mathcal{O}(\mathscr{G}) = \{t \in \Lambda(\Xi) \mid \exists u \in \mathcal{A}(\mathscr{G}).\, t = \mathscr{L}_G(u)\}$

Let us notice that since there is no structural difference between the abstract and the object vocabulary as both of them are higher-order signatures, ACGs can be combined in various ways: either by having a same abstract vocabulary shared by several ACGs in order to make two object terms (for instance a string and a logical formula) share the same underlying structure as $\mathscr{G}_{d\text{-}ed\ trees}$ and $\mathscr{G}_{Log}$ in Fig. 2, or by making the abstract vocabulary of an ACG the object vocabulary of another ACG, allowing the latter to control the admissible structures of the former, as $\mathscr{G}_{yield}$ and $\mathscr{G}_{d\text{-}ed\ trees}$ in Fig. 2.

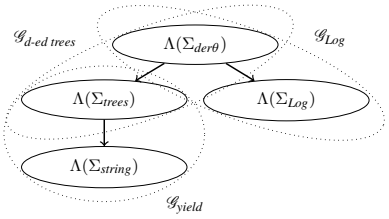


Fig. 2. ACG architecture for TAG

4 TAG as ACG

TAG derivation trees are encoded in a second order signature of an ACG [5]. An attractive side of the ACG encoding of TAG is that the grammars and the algorithms are the same for *parsing* and for *generation* [11].

A TAG derivation tree corresponds to a ground term of the second order signature. The encoding of TAG into ACG uses two ACGs $\mathscr{G}_{\textit{d-ed trees}} = \langle \Sigma_{\textit{der}\theta}, \Sigma_{\textit{trees}}, \mathscr{L}_{\textit{d-ed trees}}, \mathsf{S} \rangle$ and $\mathscr{G}_{\textit{yield}} = \langle \Sigma_{\textit{trees}}, \Sigma_{\textit{string}}, \mathscr{L}_{\textit{yield}}, \tau \rangle$. The signature $\Sigma_{\textit{der}\theta}$ is used to define derivation tree language of TAG. $\Sigma_{\textit{der}\theta}$ has constants as entries that encode elementary trees of TAG. The notion of derivation tree of TAG corresponds to the terms build on $\Sigma_{\textit{der}\theta}$ that have distinguished type S. Moreover, in order to model derived trees and surface forms, there are introduced $\Sigma_{\textit{trees}}$ and $\Sigma_{\textit{string}}$, respectively, which are defined as follows:

$\Sigma_{\textit{der}\theta}$**:** Its atomic types include S, vp, np, S_A, $\mathsf{vp}_A \ldots$ where the X types stand for the categories X of the nodes where a substitution can occur while the X_A types stand for the categories X of the nodes where an adjunction can occur. For each elementary tree $\gamma_{\textit{lex. entry}}$ it contains a constant $C_{\textit{lex. entry}}$ whose type is based on the adjunction and substitution sites as Table 1 shows. It additionally contains constants $I_X : X_A$ that are meant to provide a fake auxiliary tree on adjunction sites where no adjunction actually takes place in a TAG derivation.

$\Sigma_{\textit{trees}}$**:** Its unique atomic type is τ the type of trees. Then, for any X of arity n belonging to the ranked alphabet describing the elementary trees of the TAG, we have a constant $X_n : \overbrace{\tau \multimap \cdots \multimap \tau}^{n \text{ times}} \multimap \tau$

$\Sigma_{\textit{string}}$**:** Its unique atomic type is σ the type of strings. The constants are the terminal symbols of the TAG (with type σ), the concatenation $+ : \sigma \multimap \sigma \multimap \sigma$ and the empty string $\varepsilon : \sigma$.

Table 1 illustrates $\mathscr{L}_{\textit{d-ed trees}}$.[8] $\mathscr{L}_{\textit{yield}}$ is defined as follows:

- $\mathscr{L}_{\textit{yield}}(\tau) = \sigma$;
- for $n > 0$, $\mathscr{L}_{\textit{yield}}(X_n) = \lambda x_1 \cdots x_n . x_1 + \cdots + x_n$;
- for $n = 0$, $X_0 : \tau$ represents a terminal symbol and $\mathscr{L}_{\textit{yield}}(X_0) = X$.

We exemplify the encoding[9] of a TAG by analyzing the sentence (1), whose corresponding derivation and derived trees are given on Fig. 1.[10]

(1) *Marie* *récompense* *vraiment* *Jean*
Mary rewards really John

The term α corresponds to the derivation tree on Fig. 1; while, the image of α by $\mathscr{L}_{\textit{d-ed trees}}$ corresponds to the derived tree from Fig. 1. Finally, applying $\mathscr{L}_{\textit{yield}}$ to the derived tree produces the string representation.

$$\alpha = C_{\textit{reward}}\ I_{\mathsf{S}}\ (C^{\mathsf{V}}_{\textit{really}}\ I_{\mathsf{S}})\ C_{\textit{Marie}}\ C_{\textit{Jean}}$$
$$\mathscr{L}_{\textit{d-ed trees}}(\alpha) = \mathsf{S}_3\ (\mathsf{np}_1\ \textit{Marie})(\mathsf{v}_2\ (\mathsf{v}_1\ \textit{récompense})\ \textit{vraiment})\ (\mathsf{np}_1\ \textit{Jean})$$
$$\mathscr{L}_{\textit{yield}}(\mathscr{L}_{\textit{d-ed trees}}(\alpha)) = \textit{Marie} + \textit{récompense} + \textit{vraiment} + \textit{Jean}$$

[8] With $\mathscr{L}_{\textit{d-ed trees}}(X_A) = \tau \multimap \tau$ and for any other type X, $\mathscr{L}_{\textit{d-ed trees}}(X_A) = \tau$.
[9] We refer the reader to [14] for the details.
[10] We follow the grammar of [1].

Abstract constants $\Sigma_{der\theta}$	Their images by $\mathscr{L}_{\textit{d-ed trees}}$	The corresponding TAG trees
$C_{\textit{Jean}} : \mathsf{np}$	$c_{\textit{Jean}} \overset{:\tau}{=} \mathsf{np}_1\ \textit{Jean}$	$\gamma_{\textit{Jean}} =$ np *Jean*
$C^{\mathsf{v}}_{\textit{really}} : \mathsf{v}_A \multimap \mathsf{v}_A$	$c^{\mathsf{v}}_{\textit{really}} : (\tau \multimap \tau) \multimap (\tau \multimap \tau)$ $= \lambda^{o} vx.v\ (\mathsf{v}_2\ x\ \textit{vraiment})$	$\gamma_{\textit{really}} =$ v v* *vraiment*
$C_{\textit{reward}} : \mathsf{S}_A \multimap \mathsf{v}_A \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$	$c_{\textit{reward}} = \overset{:\ (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau}{\lambda^{o} avso.\ a\ (\mathsf{S}_3\ s\ (v\ (\mathsf{v}_1\ \textit{récompense}))\ o)}$	$\gamma_{\textit{reward}} =$ S np v np *récompense*
$I_X : X_A$	$\lambda x.x : \tau \multimap \tau$	

Table 1. TAG with Semantics as ACGs: $\mathscr{L}_{\textit{d-ed trees}}$ and $\mathscr{L}_{\textit{log.sem}}$ lexicons

5 Introduction G-TAG

G-TAG produces texts given a semantic input. In the text production task, G-TAG makes use of TAG elementary trees and G-TAG notions of *g-derivation* and *g-derived* trees.

G-Derivation Trees A TAG derivation tree can be seen as a record of the substitutions and adjunction occurring during a TAG analysis. The same is true for g-derivation tree. However, while TAG derivation trees are considered as a by-product,[11] with inflected anchors, G-TAG derivation trees are first class structures that are combined in order to reflect the conceptual input. To abstract from the surface form and from the derived tree they can relate to, they don't correspond to inflected forms but bear features that are used in a post-processing step. Complex g-derivation trees are built by going through the dynamic selection process of a lexical item from the set of appropriate candidates for a given concept. So contrary to TAG derivation trees, they are not fully instantiated trees: their arguments are represented by variables whose lexicalization are not carried out yet.

G-Derived Trees A g-derivation tree defines a unique g-derived tree corresponding to it. This correspondence is maintained all along the production process and a post-processing module outputs the surface representation (text) from the g-derived tree. In addition to inflecting forms using the feature values the post-processing module can perform some rewriting to propose *different versions* of the initial text. In this particular sense, g-derived tree corresponds to possible multiply text outputs generated by the post-processing module.

[11] Despite that TAG derivation trees were considered as good candidates for extracting semantics, it was possible to straightforwardly use them, because of which there were proposed several modifications of the formalism [15], [10].

A Conceptual Representation: G-TAG Input G-TAG conceptual representation language is a sublanguage of LOGIN defined in [2][12] G-TAG conceptual representation makes use of notions as *second order relation*, *first order relation* and *thing*. So, they are divided into two categories: RELATION and THING. Second order relations have two arguments which are from RELATION category (either first or second order ones), whereas first order relations have THING as their arguments.

On the table 2 is presented a G-TAG input that is given in [3], and which will be further used in the paper. For instance, E_0 is an input, where SUCCESSION is a second order relation having two arguments, which are relations themselves.

E_0 =: SUCCESSION [1st-EVENT ⇒ E_1, 2ND-EVENT ⇒ E_2]
E_1 =: GOAL [Action ⇒ E_{11}, Purpose ⇒ E_{12}]
E_2 =: NAPPING [NAPPER ⇒ H_1]
E_{11} =: VACUUMING [VACUUMER ⇒ H_1]
E_{12} =: REWARDING [REWARDER ⇒ H_2, REWARDEE ⇒ H_1]
H_1 =: HUMAN [NAME ⇒ "Jean", gender ⇒ masc]
H_2 =: HUMAN [NAME ⇒ "Marie", gender ⇒ fem]

Table 2. G-TAG conceptual representation input

5.1 G-TAG Generation Process

Let us assume the input of Table 2. The G-TAG process starts by lexicalizing relations that have the widest scope in the conceptual representation: typically second order relations, then first order relations, and finally, things.[13] Back to the example, we first lexicalize the second order relation of E_0: SUCCESSION. Several items are associated to this relation: après (after), avant (before), ensuite (afterwards), auparavant (beforehand), puis (then), etc. Each of them has two arguments, however, some of them produce texts comprising from two or more sentences, like ensuite(afterwards); some of them can produce either two sentence texts or one sentence text, while others produce only one sentence. For instance, *Jean a passé l'aspirateur. Ensuite, il a fait une sieste* (John has vacuumed. Afterwards, he took a nap) is a two sentence text while *John a fait une sieste après avoir passé l'aspirateur* (John took a nap after having vacuumed) is a one sentence text. For this reason, items describing the arguments or the result of second order relations have features expressing the following constraints: $(+\mathrm{T}, +\mathrm{S})$ indicates it is a text (two ore more sentences); $(+\mathrm{S})$ indicates it is either a single sentence or a text; $(-\mathrm{T}, +\mathrm{S})$ indicates it is a sentence (not a text). Every second order relation has three features: one for output, and two for inputs. [14]

[12] LOGIN is more expressive than Prolog, and at the same time as LOGIN uses unification rather than the resolution used in Prolog, LOGIN is more efficient from computational points of view.

[13] Correctness of the process is ensured because the grammars do not contain auxiliary trees that would invert the predication order. [3] argues such cases do not occur in technical texts, the first target of G-TAG. We do not elaborate on this point since the ACG approach we propose remove this constraint for free.

[14] In G-TAG, any discourse connective has exactly two arguments. A discussion about this point is provided in [3].

Let us assume that the G-TAG g-derivation tree ensuite$(+\mathrm{T}, +\mathrm{S})$ belonging to the lexical database associated with the concept SUCCESSION is chosen so that the result is a text rather than a sentence (illustrated by the leftmost g-derivation tree of Figure 3 . The process then tries to realize E_1 and E_2, the arguments of E_0. E_1 involves the GOAL relation that can be realized either by pour (in order to) or by pour que (so that), as exemplified by the rightmost g-derivation trees of Figure 3. Both have features $(-\mathrm{T}, +\mathrm{S})$ for the inputs (i.e. arguments) and return a tree labeled at the root by $(-\mathrm{T}, +\mathrm{S})$.

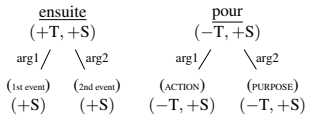


Fig. 3. G-derivation trees samples

Despite pour and pour que bear the same features, the syntactic trees corresponding to pour and pour que are quite different. For pour que S substitution nodes can be substituted by two tensed sentences, while pour takes a finite sentence and a "sentence" in the infinitive form without any nominal subject. Selecting one or the other during the generation process restricts the possible realizations for the arguments. This is enforced by a feature associated to the elementary tree, namely the $(+$reduc-subj$)$ feature as shown in Fig. 4.

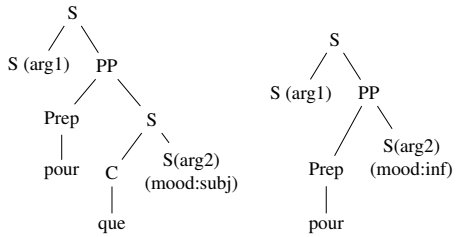


Fig. 4. TAG elementary trees corresponding *pour que* and *pour*

Again, we may assume that G-TAG selects pour, which will enforce, because of the associated elementary trees, that the subject of the first and the second arguments are the same. Afterwards, we need to lexicalize E_{11} and E_{12}. Since we chose pour, it means that E_{11} has to have H_1 as its subject that is the same subject that E_{12} has. From a semantic point of view, NAPPER (the agent) of E_{11} has to be REWARDEE (the beneficiary) of E_{12}. E_{11} can only be lexicalized as passer-l'aspirateur (run-the-vacuum-cleaner), while there are several lexicalization options for E_{12}. The REWARDING relation can be lexicalized by the following entries: récompenser (to reward), donner-récompense (to give-reward), and recevoir-récompense (to receive-reward). Let us notice that donner-récompense does not meet the constraint on a shared subject as it cannot have REWARDEE as its subject[15]. The remaining options are: recevoir-récompense, whose canonical representation has REWARDEE as the subject; and récompense whose passive construction has

[15] It lacks passivation in French and there is no form equivalent to: John was given a reward by Mary.

REWARDEE as the subject. Assuming a choice of récompenser[+passive],[16] we have lexicalized all the *relations*.

The last step of the lexicalization process, and consequently g-derivation building process is to lexicalize THING relations. So at this step, we have partially lexicalized g-derivation tree, whose the nodes that correspond to RELATION category are already lexicalized, while instead of the lexical items corresponding to *Jean* and *Marie*, H_1 and H_2 variables are used in the tree.

Let us notice that in the input (given on Table 2) H_2 occurs only once and it is in the subject position. Therefore H_2 can only be lexicalized as Marie. On the other hand, H_1 occurs three times: as arguments of E_{12} and of E_2, and as an argument of E_{11}, for which it is already lexicalized as ϵ. So, it remains to lexicalize H_1 in E_{12} and of E_2: H_1 can be either lexicalized in both of the cases as Jean, or Jean and the pronoun il (*he*). Thus, this step can be regarded as referring expression generation step. G-TAG has post-processing rules that take care of the generation of referring expressions, but we omit their formulations here. We assume that H_1 is lexicalized as Jean in E_{12} and as il in E_2. Figure 5 shows the g-derivation tree associated with the input of Table. 2 and Fig. 5 shows the unique resulting (non-inflected) derived tree as well. Afterwards, the post-processing module computes morphological information of the g-derived and outputs:

(2) *Jean a passé l'aspirateur pour être récompensé par Marie. Ensuite, il a fait une sieste.*
John vacuumed in order to be rewarded by Mary. Afterwards, he took a nap.

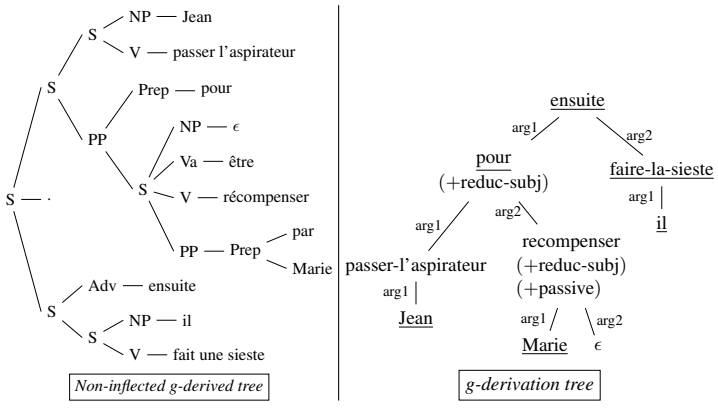


Fig. 5. g-derived and g-derivation trees of G-TAG

[16] Of course, all these branching points offer several realizations of the same entry. But for explanatory purposes, we describe only one at each step.

6 G-TAG as ACG

In order to model G-TAG in ACG, one needs to design the abstract signature $\Sigma_{\textit{g-derivations}}$ whose members will reflect the ideology of G-TAG. For instance, in G-TAG a discourse level word like *ensuite* (then), which is a lexicalization of SUCCESSION relation, can take as its arguments texts and sentences and produces text. In order to model this, we introduce types S and T. Then, we can define $D^{SS}_{\textit{then}}$: $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{T}$, which means that $D^{SS}_{\textit{then}}$ has takes two arguments of type S and returns a result of type T. As in G-TAG, *ensuite* can take two texts as arguments and return text as well, we need to do have another entry for modeling this fact. This makes us to introduce another constant $D^{TT}_{\textit{then}}$: $\mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$. For the same kind of reason, we introduce following constants: $D^{ST}_{\textit{then}}$: $\mathsf{S} \multimap \mathsf{T} \multimap \mathsf{T}$, $D^{TS}_{\textit{then}}$ and $\mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$. Other relations, like *auparavant* are modeled in the same way as *ensuite* in $\Sigma_{\textit{g-derivations}}$.

Moreover, there are other connectives such as *avant* (before) and *après* (after) that can be used in lexicalization of SUCCESSION as well, but must be modeled differently from *ensuite*. While, *ensuite* results in a text, placing side by side a text and a sentence separated with a period, in the French language, *avant* and *après* combine in a single sentence a (full) clause and an infinitive clause with an implicit subject: the one of the first clause. It is clear that in order to type *avant* and *après* in the $\Sigma_{\textit{g-derivations}}$ signature, one should use a type which schematically looks as $\ldots \multimap \mathsf{S}$. On the other hand, one needs to give the exact type to them. Despite that in TAG and G-TAG *avant* and *après* take two sentential arguments (labeled by S), the second argument bears a feature indicating it lacks the subject and that the latter has to be shared with the first sentence. For instance: *Jean a fait une sieste après avoir passé l'aspirateur* (John took a nap after having vacuumed), here the subject of *avoir passé l'aspirateur* (having vacuumed) is *Jean*, which comes from the sentence *Jean a fait une sieste* (John took a nap). So, *Jean a fait une sieste* (John took a nap) can be seen as a sentence whose subject is shared by another sentence as well. In order to model this point, we use following type: $\mathsf{Sws} \multimap \mathsf{Sh} \multimap \mathsf{np} \multimap \mathsf{S}$. Sws and Sh types correspond to the type of sentences missing a subject. Furthermore, we need to model *pour* and *pour que*, which were introduced in order to lexicalize the GOAL relation in G-TAG. The syntactic difference between *pour que* and *pour* was highlighted in Section 5.1: *pour* takes as its arguments a complete sentence and an infinitive form of a sentence missing a subject whose subject comes from the first argument. Thus, in this case, similarly to case of *avant* and *après*, *pour* has to be modeled as an entry that has type $\mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{np} \multimap \mathsf{S}$, where Sinf stands for the type of an infinitive form of a clause missing a subject; *pour que* takes as its arguments two complete (from a syntax point of view) sentences and produces a sentence, and thus can be assigned $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ type. One also needs to deal with encoding different forms of a verb. For instance, récompenser has an active and a passive form and in a G-TAG derivation tree, both of them can be encountered. That is why, two different entries need to be introduced, one for the passive form and one for the active form: $D^{\textbf{passive}}_{\textit{reward}}$ and $D^{\textbf{active}}_{\textit{reward}}$, respectively; both of them have type $\mathsf{S}_A \multimap \mathsf{v}_A \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$.

The problem of giving compositional account of v adjunction of the discourse connective was faced in G-TAG. For instance, the following text *Jean a passé l'aspirateur. Il a ensuite fait une sieste* (John vacuumed. Afterwards, he took a nap) can only be

obtained with the help of post processing module that is able to convert the sentence [*ensuite, il a fait une sieste*] into the sentence [*il a ensuite fait une sieste*]. Using ACG approach, we can handle this by introducing a constant $D^{\mathsf{v}}_{ensuite}$ of type: $\mathsf{S} \multimap (\mathsf{v}_A \multimap \mathsf{S}) \multimap \mathsf{T}$, that models the fact that $D^{\mathsf{v}}_{ensuite}$ is applied to two sentences from which one needs to take adjuntion on v.

Encoding Referring Expressions in $\Sigma_{g\text{-}derivations}$: Let us recall that in G-TAG derivation process $\mathtt{H}_1$ can be substituted by different values, like Jean, il or ϵ. In order to give account to this fact, three different constants are introduced in $\Sigma_{g\text{-}derivations}$: ${D_{Jean}}^1$, ${D_{Jean}}^2$ and ${D_{Jean}}^3$, all of which have type np. This allows us to generate terms as follows:

$$D^{SS}_{then}\ (D_{vac}\ I_{\mathsf{S}} I_{\mathsf{v}}\ {D_{Jean}}^1)\ (D_{reward}\ I_{\mathsf{S}} I_{\mathsf{v}}\ D_{marie}\ {D_{Jean}}^2)$$

There are two different constants at derivation step corresponding to *Jean*: ${D_{Jean}}^1$ and ${D_{Jean}}^2$. Thus, as already in the derivation level G-TAG has differantiated between Jean and il, we also do that by introducing different constants of the same type.

From G-TAG g-derivation Trees to TAG Derivation Trees : We translate terms of $\Sigma_{g\text{-}derivations}$, which correspond to g-derivation trees, into the TAG derivation tree language defined on $\Sigma_{der\theta}$ using the lexicon $\mathscr{L}_{der\text{-}der}$ as follows:
$\mathscr{L}_{der\text{-}der}(\mathsf{S}) = \mathscr{L}_{der\text{-}der}(\mathsf{T}) = \mathscr{L}_{der\text{-}der}(\mathsf{Sws}) = \mathscr{L}_{der\text{-}der}(\mathsf{Sinf}) = \mathscr{L}_{der\text{-}der}(\mathsf{Sh}) = \mathsf{S}$;
$\mathscr{L}_{der\text{-}der}(\mathsf{np}) = \mathsf{np}$ and $\mathscr{L}_{der\text{-}der}(I_X) = I_X$, $X = \mathsf{S}, \mathsf{v}$.

Moreover, the different constants introduced for modeling referring expressions as ${D_{Jean}}^1$, ${D_{Jean}}^2$ and ${D_{Jean}}^3$, are translated to the different constants of $\Sigma_{der\theta}$ signature:
$\mathscr{L}_{der\text{-}der}({D_{Jean}}^1) = {C_{Jean}}^1$, $\mathscr{L}_{der\text{-}der}({D_{Jean}}^2) = {C_{Jean}}^2$ and $\mathscr{L}_{der\text{-}der}({D_{Jean}}^1) = {C_{Jean}}^1$.

G-Derived Trees as Interpretation of g-Derivation Trees : As soon as g-derivation trees as terms built on $\Sigma_{g\text{-}derivations}$ are interpreted as terms built on $\Sigma_{der\theta}$, we can map them to derived trees. Thus, by composing the two lexicons $\mathscr{L}_{der\text{-}der}$ and $\mathscr{L}_{d\text{-}ed\ trees}$ we can get directly from g-derivation trees of G-TAG into TAG derived trees.

Now we can translate constants ${C_{Jean}}^1$, ${C_{Jean}}^2$ and ${C_{jean}}^3$ to different trees of Σ_{trees} from which surface realization is produced via applying $\mathscr{L}_{yield}$. For instance, in order to produce the g-derived tree from Fig. 5, we translate: $\mathscr{L}_{d\text{-}ed\ trees}({C_{Jean}}^1) = \gamma_{\textbf{John}}$, $\mathscr{L}_{d\text{-}ed\ trees}({C_{Jean}}^2) = \gamma_{il}$, and $\mathscr{L}_{d\text{-}ed\ trees}({C_{Jean}}^3) = \gamma_{\epsilon}$.

6.1 From G-TAG to Montague Style Semantics Using ACGs

In [14] is defined a signature $\Sigma_{semantics}$ and a lexicon $\mathscr{L}_{Log}$ from $\Sigma_{der\theta}$ to $\Sigma_{semantics}$. The entries in $\Sigma_{semantics}$ have Montague like semantics. Below, we sketch the signature Σ_{conrep}, where the formulas of G-TAG conceptual representations are build,[17] and the lexicon $\mathscr{L}_{der\text{-}con}$ from $\Sigma_{g\text{-}derivations}$ to Σ_{conrep}, which gives Montague style translations to the entries of $\Sigma_{g\text{-}derivations}$. In Σ_{conrep}, there are two atomic types e and t and constants such as: **j**, **m** ... of type e, the constant **VAC**, **NAP** of type $e \multimap t$; **REWARD** of type $e \multimap e \multimap t$, and **SUCC**, **GOAL** of type $t \multimap t \multimap t$.

[17] Let us notice that a formula of Σ_{conrep} signature is a higher-order logic formula that can be seen as a approximation of an conceptual representation input used in [3].

$\mathscr{L}_{der\text{-}con}$ interprets both T and S as t. np is interpreted as $[\![\mathsf{np}]\!] = (e \to t) \multimap t$, using a non-linear implication. This illustrates that semantic terms usually are not linear, but we do not elaborate on this here. [18] Typically, the sharing of the subject by two clauses related by *pour* or *avant* induces non linearity. Furthermore, Sinf, Sh, and Sws types all are interpreted as $[\![\mathsf{np}]\!] \multimap [\![\mathsf{S}]\!] = ((e \to t) \multimap t) \multimap t$ as they denote clauses lacking a subject. Then the constants of $\Sigma_{g\text{-}derivations}$ are translated as follows:

$$\mathscr{L}_{der\text{-}con}(D_{rewards}) = \lambda^{o} s\ a\ O\ S.s(S(a(\lambda x.O(\lambda y.(\mathbf{REWARD}\ x\ y))))$$
$$\mathscr{L}_{der\text{-}con}(D_{pour}) = \lambda^{o} s_1.\lambda^{o} s_2.\lambda^{o} N.N(\lambda x.(\mathbf{GOAL}(s_1(\lambda P.P\ x))(s_2(\lambda P.P\ x))))$$
$$\mathscr{L}_{der\text{-}con}(D_{apres}) = \lambda^{o} s_1.\lambda^{o} s_2.\lambda^{o} N.N(\lambda x.(\mathbf{SUCC}(s_1(\lambda P.P\ x))(s_2(\lambda P.P\ x))))$$
$$\mathscr{L}_{der\text{-}con}(D^{SS}_{then}) = \mathscr{L}_{der\text{-}con}(D^{ST}_{then}) = \cdots = \mathscr{L}_{der\text{-}con}(D^{TS}_{then}) = \lambda^{o}\ s_1 s_2.\ \mathbf{SUCC}\ s_2\ s_1$$

Encoding Referring Expressions in Σ_{conrep} : The different constants that were added to model referring expressions are translated as the same term of Σ_{conrep}. For example, we have added constants ${D_{Jean}}^{1}$, ${D_{Jean}}^{2}$ and ${D_{Jean}}^{3}$ in order to have different expressions referring to *Jean*. In other words, our aim was to produce text, where at some places instead of *Jean* there *il* (he) or for instance, nothing, i.e. ϵ (in passive sentences). Nevertheless, the conceptual representation of all these different constants ${D_{Jean}}^{i}$ for $i = 1, 2, 3$ are translated as $\lambda^{o} P.P(\mathbf{j})$, because they are different referring expressions of one and the same semantic object.

6.2 G-TAG Generation Process as ACG Morphism Composition

Given a formula F in Σ_{conrep} that corresponds to a conceptual input of G-TAG, taking into account ACG reversibility property, we can produce texts that correspond to F. This will be ACG account to G-TAG text production process that begins with accepting a conceptual representation input and ends up with outputting a corresponding surface realization. So, in order to produce a surface realization from F formula, we first have to invert the morphism $\mathscr{L}_{der\text{-}con}$, which gives the set of terms of $\Sigma_{g\text{-}derivations}$:

$${\mathscr{L}_{der\text{-}con}}^{-1}(F) = \{t_1, \ldots, t_n\}$$

Then we compute $\mathscr{L}_{der\text{-}der}(t_i)$ for $i = 1, 2, \ldots, n$. Further mapping of these terms (corresponding to g-derivation trees) $\mathscr{L}_{der\text{-}der}(t_i)$ for $i = 1, 2, \ldots, n$ via $\mathscr{L}_{d\text{-}ed\ trees}$ results in n different derived trees; after that by mapping the received derived trees via $\mathscr{L}_{yield}$, there are produced n different texts. For instance, the preimage of a formula F_0=**SUCC(VAC(j), NAP(j))** under $\mathscr{L}_{der\text{-}con}$ is the set of terms given on Fig.6. [19]

Thus, ACG composition of morphisms $\mathscr{L}_{yield}(\mathscr{L}_{d\text{-}ed\ trees}(\mathscr{L}_{der\text{-}der}\ ({\mathscr{L}_{der\text{-}con}}^{-1})))$ models G-TAG text production process within ACG framework. On the other hand, G-TAG outputs only one text, whereas using a composition of morphisms, from one formula we produce all the texts that correspond to it. This can be regarded as an advantage, as all the possible surface realizations are produced, but it has its disadvantages. Namely,

[18] It's enough to say that as long as the non-linear variables are of atomic type and are not erased, the computational complexity of inverting the morphism (ACG parsing) is unchanged.

[19] Note that there are 20 possibilities using D^{1}_{jean} and D^{2}_{jean} as lexicalizations of *Jean*.

$$
\begin{aligned}
t_1 &= D^{SS}_{\textit{then}}(D_{\textit{vac}} I_{\mathsf{S}} I_{\mathsf{V}} D^1_{\textit{jean}})(D_{\textit{sleep}} I_{\mathsf{S}} I_{\mathsf{V}} D^1_{\textit{jean}}) \\
t_2 &= D^{SS}_{\textit{bef.}}(D_{\textit{sleep}} I_{\mathsf{S}} I_{\mathsf{V}} D^1_{\textit{jean}})(D_{\textit{vac}} I_{\mathsf{S}} I_{\mathsf{V}} D^1_{\textit{jean}}) \\
t_3 &= D^{\mathsf{V}}_{\textit{ensuite}}(D_{\textit{vac}} I_{\mathsf{S}} I_{\mathsf{V}} D^1_{\textit{jean}})(\lambda^{\mathsf{o}} a. D_{\textit{sleep}}\ I_{\mathsf{S}}\ a\ D^1_{\textit{jean}}) \\
t_4 &= D_{\textit{avant}}(D^{\textit{sws}}_{\textit{vac}} I_{\mathsf{S}} I_{\mathsf{V}})(D_{\textit{sleep}} I_{\mathsf{S}} I_{\mathsf{V}}) D^1_{\textit{jean}} \\
t_5 &= D_{\textit{apres}}(D^{\textit{sws}}_{\textit{sleep}} I_{\mathsf{S}} I_{\mathsf{V}})(D_{\textit{vac}} I_{\mathsf{S}} I_{\mathsf{V}}) D^1_{\textit{jean}} \\
t_6 &= D^{SS}_{\textit{then}}(D_{\textit{vac}} I_{\mathsf{S}} I_{\mathsf{V}} D^1_{\textit{jean}})(D_{\textit{sleep}} I_{\mathsf{S}} I_{\mathsf{V}} D^2_{\textit{jean}}) \\
&\vdots \\
t_{20} &= D^{SS}_{\textit{then}}(D_{\textit{vac}} I_{\mathsf{S}} I_{\mathsf{V}} D^2_{\textit{jean}})(D_{\textit{sleep}} I_{\mathsf{S}} I_{\mathsf{V}} D^2_{\textit{jean}})
\end{aligned}
$$

Fig. 6. The Preimage of F_0 by $\mathscr{L}_{\textit{der-con}}$

among produced texts, there can be such which is comprised from valid sentences, but from discourse point of view, it is incoherent. For instance, the text produced from t_{20}, i.e. $\mathscr{L}_{\textit{yield}}(\mathscr{L}_{\textit{d-ed trees}}(\mathscr{L}_{\textit{der-der}}(t_{20})))$ is following: *Il a passé l'aspirateur. Ensuite, il a fait une sieste.* (He vacuumed. Afterwards, he took a nap.) This text (comprised from syntactically correct sentences) fails to be coherent from anaphoric reference point of view.

7 Conclusion

This paper makes shows how G-TAG can be encoded as ACG. It relies on the fact that both G-TAG and TAG encodings as ACGs make the derivation tree a primary notion. Moreover, we can benefit from the polynomial reversibility of the ACG framework. It also offers a generalization of the process to all kinds of adjunctions, including the predicative ones. As a general framework, it also allows for natural more powerful models of some linguistic attachment. This shows for instance in the compositional treatment of the V adjunction model of the *ensuite* (afterwards) adverb. The type-theoretical foundations also offers an interesting link to more evolved theories of discourse relations using, in particular higher-order types [4]. The clarification of g-derivation tree status also suggests to study in a more principled way the relations between G-TAG and other approaches of generation using TAG such as [12, 7]. Note, however, that contrary to an important part of G-TAG that offers a way (based on a semantic and linguistic analysis) to rank the different realizations of a conceptual representation, we do not deal here with such kind of preferences. As syntactic ambiguity treatment is usually not part of the syntactic formalism, we prefer the "realization ambiguity" treatment not to be part of the generation formalism. Finally, a crucial perspective is to integrate a theory of generation of referring expressions.

Acknowledgement

We would like to thank two anonymous referees for their thorough review and useful comments on the paper.

References

[1] A. Abeillé. *Une grammaire électronique du français*. Sciences du langage. CNRS Éditions, 2002.

[2] H. Aït-Kaci and R. Nasr. Login: A logic programming language with built-in inheritance. *J. Log. Program.*, 3(3):185–215, 1986.

[3] L. Danlos. G-tag: A lexicalized formalism for text generation inspired by tree adjoining grammar. In A. Abeillé and O. Rambow, editors, *Tree Adjoining Grammars: Formalisms, Linguistic Analysis, and Processing*, pages 343–370. CSLI Publications, 2000.

[4] L. Danlos. D-STAG: a formalism for discourse analysis based on sdrt and using synchronous TAG. In P. de Groote, M. Egg, and L. Kallmeyer, editors, *14th conference on Formal Grammar - FG 2009*, volume 5591 of *LNCS/LNAI*, pages 64–84. Springer, 2011.

[5] P. de Groote. Tree-adjoining grammars as abstract categorial grammars. In *Proceedings of TAG+6*, pages 145–150, 2002.

[6] P. de Groote and S. Pogodalla. On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4):421–438, 2004.

[7] C. Gardent and L. Perez-Beltrachini. RTG based surface realisation for TAG. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 367–375, Beijing, China, August 2010. Coling 2010 Organizing Committee.

[8] A. K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163, 1975.

[9] A. K. Joshi and Y. Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, volume 3, chapter 2. Springer, 1997.

[10] L. Kallmeyer. Using an enriched tag derivation structure as basis for semantics. In *Proceedings of TAG+6*, 2002.

[11] M. Kanazawa. Parsing and generation as datalog queries. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 176–183, 2007. http://www.aclweb.org/anthology/P/P07/P07-1023.

[12] A. Koller and K. Striegnitz. Generation as dependency parsing. In *ACL*, pages 17–24, 2002.

[13] F. Meunier. *Implantation du formalisme de génération G-TAG*. PhD thesis, Université Paris 7 — Denis Diderot, 1997.

[14] S. Pogodalla. Advances in Abstract Categorial Grammars: Language Theory and Linguistic Modeling. ESSLLI 2009 Lecture Notes, Part II. http://hal.inria.fr/hal-00749297, 2009.

[15] O. Rambow, K. Vijay-Shanker, and D. Weir. D-Substitution Grammars. *Computational Linguistics*, 2001.